(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 190 304 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
19.02.2003 Bulletin 2003/08

(21) Application number: **00930244.9**

(22) Date of filing: **01.05.2000**

(51) Int Cl.[7]: **G06F 9/00**

(86) International application number:
PCT/US00/11671

(87) International publication number:
WO 00/067114 (09.11.2000 Gazette 2000/45)

(54) SYSTEM AND METHOD FOR DISCOVERING AND BINDING A PROGRAM OBJECT

SYSTEM UND METHODE ZUM AUFFINDEN UND VERBINDEN EINES PROGRAMMOBJEKTES

DISPOSITIF ET TECHNIQUE DE DECOUVERTE D'UN PROGRAMME OBJET ET D'EDITION DE LIENS

(84) Designated Contracting States:
DE FR GB IE

(30) Priority: **03.05.1999 US 304429**

(43) Date of publication of application:
27.03.2002 Bulletin 2002/13

(73) Proprietor: **SUN MICROSYSTEMS, INC.**
Palo Alto, California 94303 (US)

(72) Inventor: **GUINAN, Daniel**
Redwood Shores, CA 94065 (US)

(74) Representative: **Hanna, Peter William Derek**
Peter Hanna Associates
11 Mespil Road
Dublin 4 (IE)

(56) References cited:

- **PASCAL LEDRU: "Adaptative Parallelism: An Early Experiment with Java(tm) Remote Method Invocation" OPERATING SYSTEMS REVIEW (SIGOPS),US,ACM HEADQUARTER. NEW YORK, vol. 31, no. 4, 1 October 1997 (1997-10-01), pages 24-29, XP000738267**
- **"ALGORITHM FOR LOCATING OBJECT HANDLERS IN A DISTRIBUTED SYSTEM" IBM TECHNICAL DISCLOSURE BULLETIN,US,IBM CORP. NEW YORK, vol. 36, no. 1, 1993, pages 484-485, XP000333921 ISSN: 0018-8689**
- **ANDERS KRISTENSEN, COLIN LOW: "Problem-Oriented Object Memory: Customizing Consistency" ACM SIGPLAN NOTICES,US,ASSOCIATION FOR COMPUTING MACHINERY, NEW YORK, vol. 30, no. 10, 1 October 1995 (1995-10-01), pages 399-413, XP000537918 ISSN: 0362-1340**
- **YASUHIKO YOKOTE, FUMIO TERAOKA, ATSUSHI MITSUZAWA, NOBUHISA FUJINAMI, MARIO TOKORO: "The muse Object Architecture: A New Operating System Structuring Concept" OPERATING SYSTEMS REVIEW (SIGOPS),US,ACM HEADQUARTER. NEW YORK, vol. 25, no. 2, 1 April 1991 (1991-04-01), pages 22-46, XP000297108**
- **C. F. CODELLA, D. N. DILLENBERGER, R. D. JACKSON, T. A. MIKALSEN, I. SILVA-LEPE: "Support for Enterprise JavaBeans in Component Broker" IBM SYSTEMS JOURNAL,US,IBM CORP. ARMONK, NEW YORK, vol. 37, no. 4, 1998, pages 502-537, XP000805714 ISSN: 0018-8670**

EP 1 190 304 B1

place of the interface or interface service requested by the program.

[0011] Instead of identifying an implementation class or instance of an implementation, the registry may identify a class loader associated with an implementation of the interface or information from which such a class loader may be generated. In this case, the class loader is instantiated (unless, perhaps, an instance of the class loader is already available), and the implementation is then instantiated. A reference to the newly created implementation instance is then returned to the program or the implementation may be invoked in place of the interface or interface service requested by the program.

[0012] However, the registry may not identify an implementation of the interface, an associated class loader or means for creating one or the other. In this event a locator attempts to locate such information so that, for example, the system can create a class loader that can instantiate an implementation of the interface. The locator may be configured to search pre-specified areas (e.g., local directories and/or remote servers), dynamically specified areas (e.g., depending upon the program from which the interface is invoked, the type of interface or an identity of a user) or all accessible areas. In particular, the locator is designed to search across a network such as the Internet. If suitable information is found, it can be stored in the registry for easy location and retrieval the next time it is needed. The procedure described above for creating a class loader and instantiating an implementation is then performed.

[0013] One embodiment of the invention thus provides a method of constructing a self-organizing software environment in which objects (e.g., interface implementations) are located and bound at the time a particular service, function or behavior is required. If one object retrieved through this method requires further support (e.g., a service of another interface), another object may be located and attached to the environment when needed. Thus, functionality may be added to a particular program that was not envisioned when the program was developed.

BRIEF DESCRIPTION OF THE FIGURES

[0014]

FIG. 1 is a block diagram depicting a system for resolving a request for an implementation of an interface in accordance with an embodiment of the present invention.

FIG. 2 is a flow chart demonstrating a method of resolving a request for an implementation of an interface in accordance with an embodiment of the present invention.

FIG. 3 is a flow chart demonstrating an alternative method of resolving a request for an implementation of an interface in accordance with an embodiment of the present invention.

FIG. 4 is a block diagram of a hierarchical context framework for facilitating the resolution of a request for an interface in accordance with an embodiment of the invention.

DETAILED DESCRIPTION

[0015] The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of particular applications of the invention and their requirements. In particular, one skilled in the art will recognize that the present invention is not limited to the use of the Java™ programming language. The use of other object-oriented programming languages are similarly envisioned. Thus, the present invention is not intended to be limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

[0016] A program environment in which a present embodiment of the invention is executed may incorporate a general-purpose computer or a special purpose device such a hand-held computer. Details of such devices (e.g., processor, memory, data storage and display) are well known and are omitted for the sake of clarity. It should also be understood that the techniques of the present invention might be implemented using a variety of technologies. For example, the methods described herein may be implemented in software running on a computer system, or implemented in hardware utilizing either a combination of microprocessors or other specially designed application specific integrated circuits, programmable logic devices, or various combinations thereof. In particular, methods described herein may be implemented by a series of computer-executable instructions residing on or in a storage medium such as a carrier wave, disk drive, or other computer-readable medium. Exemplary forms of carrier waves may take the form of electrical, electromagnetic or optical signals conveying digital data streams along a local network or a publicly accessible network such as the Internet.

[0017] In a present embodiment of the invention a system and methods are provided for resolving a request for, or invocation of, an interface (e.g., an application programming interface, or API) or an implementation of an interface. A program may request an implementation of an interface, for example, in order to access a service or method offered or identified by the interface. In an object-oriented programming environment such as Java™, an interface may be viewed as a set of abstract base classes containing one or more services or methods that may be used by a program. Illustratively, an interface in the Java™ programming language does not, however, include service definitions (e.g., executable code for performing the services). In the Java™ programming language, the implementation of an interface provides the necessary code for performing the interface's service(s).

LAN or WAN or a public network such as the Internet, or other communication link.

[0026] Also, the search process may comprise multiple steps. Locator 104 may, for example, first search for a package of code containing a suitable implementation of the interface. Once such a package is found, locator 104 may then search the package of code for the suitable implementation.

[0027] If locator 104 is successful, it passes the information it locates to resolver 106, which operates as described above. Locator 104 may, for example, find a principal concerning a class loader that can load a suitable implementation, in which case resolver 106 then instantiates the class loader and the interface implementation, as necessary.

[0028] Illustratively, a principal describes how to create the object (i.e., implementation) that is to be bound to a requesting program. Thus, a principal may contain information identifying how to create a class loader, such as the location of program code to download and execute, instructions for extracting the class loader from the program code, etc. In particular, in a Java™ programming environment a principal may be a URL (Uniform Resource Locator) or other URI (Uniform Resource Identifier) of a piece of Java™ code, a JAR (Java™ ARchive) file, instructions for creating a class loader (e.g., a script, a Java™ class, a set of rules), etc. A principal may also store the class name of an implementation (and/or its associated interface) that can be created once the principal's class loader is instantiated. Thus, by searching for the name of an implementation (which may be similar to its associated interface), an interface or interface service, a suitable principal for creating a class loader and instantiating the implementation can be found.

[0029] If locator 104 cannot locate an implementation of the interface or a class loader capable of creating an instance of an implementation of the interface, it may attempt to satisfy the program request in an alternative manner. In one embodiment of the invention, locator 104 may generate or compose an artificial or "dummy" implementation. Alternatively, locator 104 returns an exception (e.g., an error message) to program code 100.

[0030] One skilled in the art will appreciate that the division of labor among registry 102, locator 104 and resolver 106 is flexible and is not limited to the arrangement described above. In particular, FIG. 1 depicts just one system for locating a suitable implementation of a requested interface and returning a reference to an instance of the implementation. In an alternative embodiment of the invention, registry 102 may be configured to act primarily as a data repository to store information aiding a search for an implementation of the requested interface. The functions of locating an implementation of an interface, whether registered or not, and resolving it to an instance of a suitable implementation are then performed by one or more modules acting with varying degrees of similarity to locator 104 and/or resolver 106.

[0031] Registry 102 may comprise data relating to all interfaces that have been requested or invoked or, for example, some number of the most recently or most frequently requested interfaces. A portion of registry 102 may, for example, comprise a repository (e.g., a cache) having a relatively fast response time. Such a repository would be well suited to storing the most recently or frequently used interfaces.

[0032] In addition, after locator 104 locates information for an implementation not known to registry 102, information concerning the implementation or its class loader may be stored in registry 102 for easier retrieval the next time the same interface is requested.

[0033] In different embodiments of the invention the operation and management of registry 102, locator 104 and resolver 106 may be performed by different entities. In particular, in one embodiment of the invention a Java™ Virtual Machine operates and manages one or more of these modules on behalf of one or more application programs comprising program code 100. In an alternative embodiment, however, an application program comprising 100 may incorporate the ability to maintain and operate registry 102, locator 104 and resolver 106.

[0034] One skilled in the art will appreciate that within a given programming environment, numerous methods of locating or creating a suitable implementation of a particular interface or interface service may be constructed. Likewise, many methods of resolving implementations with requested interfaces, and registering or tracking such resolutions may be designed.

[0035] Operation of the system depicted in FIG. 1 may be recursive in that an implementation that is located and bound in response to the invocation of an interface or service by program code 100 may, in turn, request another interface or interface service. In particular, a first implementation that is resolved by resolver 106 may specify a number of interfaces or interface services that it requires. The procedure described above is thus performed again to locate and resolve other implementations as needed. Thus, a sort of self-organizing software environment may be constructed in which interface implementations are located and "attached" to the environment when an interface is invoked or requested. As a result, program code 100 may gain the ability to perform tasks not envisioned by its developer.

[0036] U.S. Patent No. 5,727,147 (the '147 patent), issued March 10, 1998 and entitled "System and Method for Resolving Symbolic References to Externally Located Program Files" addresses the creation of application-specific class loaders. In the '147 patent an application-specific class loader or an object class instantiated by an application-specific class loader may be located on a remote computer. The location of the remote object(s) however, are known beforehand. Thus, the '147 patent does not describe a search procedure such as described above for the embodiment of the present invention depicted in FIG. 1. In addition, the '147 patent

(e.g., the default set of locations to be searched for a Java™ class, object or set of code), the classpath of the class loader associated with the requested interface, a remote server or system that hosts the requesting program, a central server, a library server, a network search utility (e.g., Yahoo, Excite), an Internet site, etc.

[0047] If in state 210 the search for a class loader or principal is unsuccessful, an exception (e.g., an error message) is returned to the requesting program in state 212. In an alternative embodiment of the invention, the system performing the illustrated method may attempt to determine some other manner of performing a requested service for which no implementation can be located, unless the service is part of a critical operation of the requesting program. The illustrated method then ends with end state 220.

[0048] If, however, an entry was found in the registry for the requested interface (state 206), or a class loader or principal was found pursuant to a search (state 210), the illustrated method proceeds to state 214. In state 214 a reference to a suitable class loader is retrieved (e.g., from the registry) or a class loader associated with an implementation of the interface is created (e.g., instantiated) from the available information.

[0049] In state 216 an instance of the implementation is retrieved, if it exists, or the class loader is called upon to instantiate the implementation class. In one embodiment of the invention, if an interface is found to be registered in state 206 the illustrated method may proceed directly to state 216 from state 206. This may occur, for example, if the registry directly identifies an existing instance of the implementation.

[0050] In state 218, the implementation is invoked in place of the interface invocation received in state 202. After state 218 the method ends with end state 220.

[0051] FIG. 3 depicts one alternative method of satisfying a program's request for an interface. In this method the program framework itself incorporates the ability to locate a suitable implementation of the requested interface. As one skilled in the art will recognize, the portion of the program that performs the illustrated method may require certain privileges in order to perform the identified tasks. State 300 is a start state.

[0052] In state 302, a component (e.g., object) of an application program requires an interface or interface service. The component may invoke the interface or service during its execution or the application framework may identify the need for the interface before the component makes the invocation. The application framework may, for example, identify the need for the interface when the program component is instantiated.

[0053] In state 304 a registry is searched, illustratively using the requested interface as a key or index. As in the method of FIG. 2, for each interface registered in the registry a class loader that can install or create an implementation of the interface may be identified. The registry may store a reference to a class loader instance or, alternatively, identify information from which a class loader capable of creating an implementation of the interface may be created (e.g., a principal).

[0054] If, in state 306, no entry is found in the registry for the requested interface, in state 308 a search is conducted for a class loader of an implementation of the interface and/or a principal from which such a class loader may be created.

[0055] If in state 310 the search for a class loader or principal is unsuccessful, an exception (e.g., an error message) is returned to the requesting program in state 312. In an alternative embodiment of the invention the application framework may attempt to determine some other manner of performing a requested service for which no implementation can be located, unless the service is part of a critical operation of the program. The illustrated method then ends with end state 320.

[0056] If, however, an entry was found in the registry for the requested interface (state 306), or a class loader or principal was found pursuant to a search (state 310), the illustrated method proceeds to state 314. In state 314 a reference to a suitable class loader is retrieved (e.g., from the registry) or a class loader associated with an implementation of the interface is created (e.g., instantiated) from the available information.

[0057] In state 316 an instance of the implementation is retrieved, if it exists, or the class loader is called upon to instantiate the implementation class. In one embodiment of the invention, if an interface is found to be registered in state 306 the illustrated method may proceed directly to state 316 from state 306.

[0058] In state 318, a reference to an instance of the implementation is returned to the program. After state 318 the method ends with end state 320. The program may invoke the implementation as part of end state 320.

[0059] In one alternative embodiment of the invention, a registry contains information concerning implementations of interfaces in addition to or in place of class loader and/or principal information. The registry may, for example, comprise multiple data structures or repositories (e.g., databases, tables, arrays, flat files, lists). A first portion of a registry may comprise a relatively fast data structure (e.g., an object table, a hardware or software cache) consisting of entries mapping interfaces to classes of implementations of the interface or instances of such implementations. Entries included in this first portion may be limited to the most recently or most frequently used interfaces.

[0060] When an implementation class for a requested interface is identified in this first portion of a registry, the implementation is instantiated (e.g., by resolver 106 of FIG. 1). Instead of identifying an implementation class, however, the registry may identify a reference to an existing instance of the implementation. The implementation is then returned to the requesting program (e.g., by returning a reference to a new or existing instance) or is invoked in place of the interface invocation. If there is no entry for a requested interface in a first portion of the registry in this alternative embodiment, a second portion

within the framework (e.g., at a low level number) may export its services to services lower in the framework (e. g.. at a higher-numbered level) by storing appropriate information in the contexts associated with the lower services. Thus, when an object or application in context 442 requires a particular service, the registry for context 442 may contain a reference to a definition of the service (e.g., an instance of an implementation of the service's interface) in context 420, for example. This avoids having to pass the request through the framework to context 430 and then context 420.

[0073] As specified above, within a framework of contexts for resolving requests for interfaces or interface services each context may maintain its own registry. Thus, users connected to different contexts may have their requests for the same interface satisfied by different implementations of that interface. This may be illustrated by revisiting the adventure game example described above. One set of players (e.g., adults) may be associated with a context in which Monster objects are imbued with menacing attributes. A Dinosaur Monster, for example, may be an enormous carnivore intent on devouring a player's character. Another set of players (e g children) may be associated with a context in which Monster objects are more innocuous. A Dinosaur Monster in this context, for example, may be purple and sport a smarmy façade (but still exhibit noisome activity such as singing).

[0074] Thus. the context with which a user is associated may define the behavior of a program operated by the user The context's registry may be maintained by a portion of an application framework associated with the context by a JVM or by some other entity with suitable privileges or access to locations at which interface implementations may be stored. Through its registry, a context may inform an application program framework or JVM where to look for certain objects, how to find a definition for an interface service, what objects should be connected to a user's program, etc.

[0075] Each context's registry may be created when the context is spawned or when a first user is placed in the context. Further, each context's registry may be initialized by copying the registry of its parent. By copying registries. the export of services from more general to more specific contexts is simplified. Within a context, however. its registry is updated as interfaces are requested and implementations of the requested interfaces are located and resolved.

[0076] The foregoing descriptions of embodiments of the invention have been presented for purposes of illustration and description only. They are not intended to.be exhaustive or to limit the invention to the forms disclosed. Many modifications and variations will be apparent to practitioners skilled in the art. Accordingly, the above disclosure is not intended to limit the invention; the scope of the invention is defined by the appended claims.

[0077] In particular, in one or more of the embodiments of the invention described above, a registry may be supplemented over time as requests for interfaces are satisfied. For example, as described in conjunction with the method illustrated in FIG. 2, a search is made for a class loader or principal when a requested interface is unknown to the registry. If such a search is successful, an entry is made in the registry so that a subsequent request for the same interface may be satisfied without the delay that is inherent in having to conduct a search.

Claims

1. A method of resolving a first reference to a first interface service issued by a program executing on a computer, the method comprising:

   receiving (202) a first reference to a first interface service from a program executing on a computer;
   searching (204) for a program object class that is configured to define said first interface service;
   identifying (216) a first program object class unknown to said program at the time said first reference is received;
   if an instance of said first object class is unavailable, creating (216) an instance of said first object class; and
   resolving (218) said first reference with said instance of said first object class.

2. The method of claim 1, further comprising identifying (214) a class loader configured to instantiate said first object class.

3. The method of claim 1, wherein said searching (204) comprises determining whether an object class configured to define said first service is identified in a first memory of the computer.

4. The method of claim 1, wherein said searching (204) comprises searching across a publicly accessible network for an object class configured to define said first interface service.

5. The method of claim 1, wherein said resolving (218) comprises binding said instance of said first object class to said program.

6. The method of claim 1, further comprising determining whether said instance of said first object class invokes a second interface service unknown at the time the program was loaded.

7. The method of claim 1, wherein said searching (204) comprises:

terface service.

19. The method of claim 17, further comprising:

receiving a second reference to a second interface service during execution of said instance of said first implementation of said first interface service;
discovering (308) a class loader configured to instantiate an implementation of said second interface service;
resolving (318) said second reference by loading an instance of said implementation of said second interface service; and
adding to said first memory information concerning said implementation of said second interface service.

20. The method of claim 17, wherein said discovering (314) comprises searching one or more electronic storage areas for an implementation of said first information service that is not already identified in said first memory.

21. The method of claim 20, wherein one of said one or more electronic storage areas is remote from the computer.

22. The method of claim 20, wherein said discovering further comprises:

locating said first implementation of said first interface service; and
instantiating (316) said first implementation of said first interface service.

23. The method of claim 20, wherein said discovering further comprises:

if said searching for an implementation of said first interface service is unsuccessful, searching (308) for means of facilitating the creation of an implementation of said first interface service not already identified in said first memory.

24. The method of claim 23, wherein said means is a class loader.

25. The method of claim 23, wherein said discovering further comprises:

if said searching for means of creating an implementation of said first interface service is unsuccessful, searching for means of facilitating the creation of a class loader configured to load an implementation of said first interface service that is not already identified in said first memory.

26. The method of claim 25, wherein said means is a Uniform Resource Identifier, a Java Archive file, or a series of computer-executable instructions for creating a class loader.

27. A computer system comprising:

a processor configured to execute a program (100), wherein during said execution of the program a first reference is made to a first interface service, and wherein said reference must be resolved in order to continue said execution;
a registry (102) configured to identify means for resolving references to interface services made during execution of the program;
a locator (104) configured to search said registry for means of resolving said first reference; and
a resolver (106) configured to access a first implementation of said first interface service and resolve said first reference with an instance of said first implementation.

28. The computer system of claim 27, wherein said first implementation was unknown at the time the program was loaded for execution by said processor.

29. The computer system of claim 27, wherein said locator (104) is further configured to search one or more electronic storage areas other than said registry for means of resolving said first reference.

30. The computer system of claim 27, wherein said locator (104) is further configured to search one or more electronic storage areas remote to the computer system for means of resolving said first reference.

31. The computer system of claim 27, wherein said registry (102) comprises a memory configured to store an identifier of a known implementation of an interface service.

32. The computer system of claim 27, wherein said registry (102) comprises a memory configured to store an identifier of a class loader with which an implementation of an interface service may be created.

33. The computer system of claim 27, wherein said registry (102) includes a hierarchical set (400) of contexts, said hierarchical set of contexts comprising:

a first context (410), comprising a first definition of an interface service; and
a second context (420), comprising a second definition of another interface service;

wherein said second context is searched for

11. Verfahren nach Anspruch 10, bei dem der Versuch (208) einer Lokalisierung umfaßt:

Suchen eines oder mehrerer Kontexte in einer hierarchischen Menge (400) von Kontexten, wobei jeder Kontext in der Menge von Kontexten so konfiguriert ist, daß er eine Bezugnahme auf einen Schnittstellendienst für eine den Dienst definierende Implementierung auflöst; und
falls die erste Bezugnahme in einem ersten Kontext (420) in der hierarchischen Menge von Kontexten nicht aufgelöst werden kann, Versuchen, die erste Bezugnahme in einem zweiten Kontext (410) in der hierarchischen Menge von Kontexten aufzulösen.

12. Verfahren nach Anspruch 1, bei dem das Auflösen (218) die Rückführung (318) einer Bezugnahme auf die Instanz der ersten Objektklasse zum Programm umfaßt.

13. Verfahren nach Anspruch 1, bei dem das Auflösen (218) das Aufrufen (218) der Instanz der ersten Objektklasse umfaßt.

14. Verfahren nach Anspruch 1, bei dem das Suchen (204) das Durchsuchen eines ersten Speichers nach einer Implementierung des ersten Schnittstellendienstes umfaßt, wobei der erste Speicher in dem Computer vorhanden ist.

15. Verfahren nach Anspruch 14, bei dem das Suchen (204) ferner das Suchen eines entfernten Computersystems, das mit dem Computer elektrisch gekoppelt ist, umfaßt.

16. Verfahren nach Anspruch 1, bei dem das Suchen (204) das Identifizieren (214) eines Klassenladers, der einer Implementierung des ersten Schnittstellendienstes zugeordnet ist, umfaßt.

17. Verfahren zum Erweitern einer dynamischen Sammlung von Informationen, um eine Schnittstellendienst-Bezugnahme von einem Programm für eine Implementierung des Schnittstellendienstes, auf den Bezug genommen wird, aufzulösen, wobei die dynamische Sammlung von Informationen so konfiguriert ist, daß sie Informationen enthält, die Implementierungen betreffen, die zu dem Zeitpunkt, zu dem das Programm geladen wird, bekannt oder unbekannt sind, wobei das Verfahren umfaßt:

Laden eines Programms in einen Computer, wobei während des Ladens eine oder mehrere bekannte Implementierungen von Schnittstellendiensten, auf die durch das Programm Bezug genommen werden kann, in einem ersten Speicher identifiziert werden;
Empfangen (302) einer ersten Bezugnahme auf einen ersten Schnittstellendienst während der Ausführung des Programms;
Finden (314) einer ersten Implementierung des ersten Schnittstellendienstes, wobei die erste Implementierung in dem ersten Speicher nicht identifiziert wird;
Hinzufügen von Informationen, die die erste Implementierung des ersten Schnittstellendienstes betreffen, zu dem ersten Speicher; und
Auflösen (318) der ersten Bezugnahme durch Binden einer Instanz der ersten Implementierung des ersten Schnittstellendienstes.

18. Verfahren nach Anspruch 17, das ferner umfaßt:

Empfangen einer zweiten Bezugnahme auf einen zweiten Schnittstellendienst während der Ausführung der Instanz der ersten Implementierung des ersten Schnittstellendiensts;
Finden (314) einer Implementierung des zweiten Schnittstellendiensts, die in dem ersten Speicher nicht identifiziert wird;
Auflösen (318) der zweiten Bezugnahme durch Binden einer Instanz der Implementierung des zweiten Schnittstellendiensts; und
Hinzufügen von Informationen, die die Implementierung des zweiten Schnittstellendiensts betreffen, zu dem ersten Speicher.

19. Verfahren nach Anspruch 17, das ferner umfaßt:

Empfangen einer zweiten Bezugnahme auf einen zweiten Schnittstellendienst während der Ausführung der Instanz der ersten Implementierung des ersten Schnittstellendiensts;
Finden (308) eines Klassenladers, der so konfiguriert ist, daß er eine Implementierung des zweiten Schnittstellendiensts instanziiert;
Auflösen (318) der zweiten Bezugnahme durch Laden einer Instanz der Implementierung des zweiten Schnittstellendiensts; und
Hinzufügen von Informationen, die die Implementierung des zweiten Schnittstellendiensts betreffen, zum ersten Speicher.

20. Verfahren nach Anspruch 17, bei dem das Entdekken (314) das Durchsuchen eines oder mehrerer elektronischer Speicherbereiche nach einer Implementierung des ersten Informationsdiensts, der nicht bereits in dem ersten Speicher identifiziert worden ist, umfaßt.

21. Verfahren nach Anspruch 20, bei dem einer der mehreren elektronischen Speicherbereiche sich entfernt von dem Computer befindet.

terprogramms unbekannt war, nach einem der Ansprüche 1 bis 16 oder einem der Ansprüche 17 bis 26 auszuführen.

35. Computerprogramm nach Anspruch 34, das als ein computerlesbares Speichermedium ausgeführt ist.

Revendications

1. Procédé pour résoudre une première référence à un premier service d'interface envoyée par un programme s'exécutant sur un ordinateur, le procédé comportant les étapes consistant à :

   recevoir (202) une première référence à un premier service d'interface provenant d'un programme s'exécutant sur un ordinateur,
   rechercher (204) une classe d'objet de programme qui est configurée pour définir ledit premier service d'interface,
   identifier (216) une première classe d'objet de programme inconnue dudit programme au moment où ladite première référence est reçue,
   si une instance de ladite première classe d'objet n'est pas disponible, créer (216) une instance de ladite première classe d'objet, et
   résoudre (218) ladite première référence à l'aide de ladite instance de ladite première classe d'objet.

2. Procédé selon la revendication 1, comportant en outre l'identification (214) d'un chargeur de classe configuré pour instancier ladite première classe d'objet.

3. Procédé selon la revendication 1, dans lequel ladite recherche (204) comporte la détermination du fait qu'une classe d'objet configurée pour définir ledit premier service est identifiée dans une première mémoire de l'ordinateur.

4. Procédé selon la revendication 1, dans lequel ladite recherche (204) comporte la recherche à travers un réseau accessible publiquement d'une classe d'objet configurée pour définir ledit premier service d'interface.

5. Procédé selon la revendication 1, dans lequel ladite résolution (218) comporte l'édition de lien de ladite instance de ladite première classe d'objet audit programme.

6. Procédé selon la revendication 1, comportant en outre l'étape consistant à déterminer si ladite instance de ladite première classe d'objet appelle un second service d'interface inconnu au moment où le programme a été chargé.

7. Procédé selon la revendication 1, dans lequel ladite recherche (204) comporte les étapes consistant à :

   chercher un ou plusieurs contextes dans un ensemble hiérarchique (400) de contextes, où chaque contexte dudit ensemble de contextes est configuré pour résoudre une référence à un service d'interface relatif à un objet configuré pour définir le service d'interface, et
   tenter de résoudre ladite première référence dans un second contexte (410) si ladite première référence ne peut être résolue dans un premier contexte (420).

8. Procédé selon la revendication 1, comportant en outre les étapes consistant à :

   recevoir une seconde référence à un second service d'interface pendant ladite résolution de ladite première référence, où ledit second service d'interface était inconnu avant ladite résolution, et
   accéder à une instance d'une première implémentation dudit second service d'interface.

9. Procédé selon la revendication 8, comportant en outre un enregistrement de ladite première instance de ladite première classe d'objet et de ladite première implémentation dudit second service d'interface dans une structure de données configurée pour faciliter la résolution de références de programme.

10. Procédé selon la revendication 1, dans lequel ladite recherche (204) comporte les étapes consistant à :

    tenter de localiser une implémentation de ladite première classe d'objet de programme,
    si ladite tentative pour localiser une implémentation de ladite première classe d'objet de programme échoue, tenter (208) de localiser un chargeur de classe configuré pour créer une implémentation de ladite première classe d'objet de programme, et
    si ladite tentative pour localiser un chargeur de classe échoue, tenter (208) de localiser des moyens pour créer un chargeur de classe configuré pour créer une implémentation de ladite première classe d'objet de programme.

11. Procédé selon la revendication 10, dans lequel ladite tentative (208) de localisation comporte les étapes consistant à :

    chercher un ou plusieurs contextes dans un ensemble hiérarchique (400) de contextes, où chaque contexte dudit ensemble de contextes est configuré pour résoudre une référence à un

premier service d'interface, et
instancier (316) ladite première implémentation dudit premier service d'interface.

23. Procédé selon la revendication 20, dans lequel ladite découverte comporte en outre l'étape consistant à :

si ladite recherche d'une implémentation dudit premier service d'interface échoue, rechercher (308) des moyens pour faciliter la création d'une implémentation dudit premier service d'interface non déjà identifiée dans ladite première mémoire.

24. Procédé selon la revendication 23, dans lequel lesdits moyens sont un chargeur de classe.

25. Procédé selon la revendication 23, dans lequel ladite découverte comporte en outre l'étape consistant à :

si ladite recherche de moyens de création d'une implémentation dudit premier service d'interface échoue, rechercher des moyens pour faciliter la création d'un chargeur de classe configuré pour charger une implémentation dudit premier service d'interface qui n'est pas déjà identifiée dans ladite première mémoire.

26. Procédé selon la revendication 25, dans lequel lesdits moyens sont un Identificateur de Ressource Uniforme, un fichier d'Archives Java, ou une série d'instructions exécutables par ordinateur destinés à créer un chargeur de classe.

27. Système informatique comportant :

un processeur configuré pour exécuter un programme (100), où pendant ladite exécution du programme, une première référence est effectuée à un premier service d'interface, et où ladite référence doit être résolue afin de continuer ladite exécution,
un registre (102) configuré pour identifier des moyens destinés à résoudre des références à des services d'interface effectués pendant l'exécution du programme,
un localisateur (104) configuré pour rechercher dans ledit registre des moyens destinés à résoudre ladite première référence, et
un dispositif de résolution (106) configuré pour accéder à une première implémentation dudit premier service d'interface et résoudre ladite première référence à l'aide d'une instance de ladite première implémentation.

28. Système informatique selon la revendication 27, dans lequel ladite première implémentation était inconnue au moment où le programme a été chargé pour être exécuté par ledit processeur.

29. Système informatique selon la revendication 27, dans lequel ledit localisateur (104) est en outre configuré pour rechercher dans une ou plusieurs zones de mémorisation électroniques autres que ledit registre des moyens de résolution de ladite première référence.

30. Système informatique selon la revendication 27, dans lequel ledit localisateur (104) est en outre configuré pour rechercher dans une ou plusieurs zones de mémorisation électroniques distantes du système informatique des moyens de résolution de ladite première référence.

31. Système informatique selon la revendication 27, dans lequel ledit registre (102) comporte une mémoire configurée pour mémoriser un identificateur d'une implémentation connue d'un service d'interface.

32. Système informatique selon la revendication 27, dans lequel ledit registre (102) comporte une mémoire configurée pour mémoriser un identificateur d'un chargeur de classe à l'aide duquel une implémentation d'un service d'interface peut être créée.

33. Système informatique selon la revendication 27, dans lequel ledit registre (102) inclut un ensemble hiérarchique (400) de contextes, ledit ensemble hiérarchique de contextes comportant :

un premier contexte (410), comportant une première définition d'un service d'interface, et
un second contexte (420), comportant une seconde définition d'un autre service d'interface,

dans lequel ledit second contexte fait l'objet d'une recherche d'une définition dudit premier service d'interface et, si ledit second contexte ne contient pas ladite définition, ledit premier contexte est recherché.

34. Programme informatique qui, lorsqu'il est exécuté par un ordinateur, amène l'ordinateur à exécuter un procédé pour résoudre une référence provenant d'un programme informatique à un service d'interface à l'aide d'une implémentation de l'interface qui était inconnue pendant le chargement du programme informatique, selon l'une quelconque des revendications 1 à 16, ou des revendications 17 à 26.

35. Programme informatique selon la revendication 34, mis en oeuvre sur un support de mémorisation lisible par ordinateur.

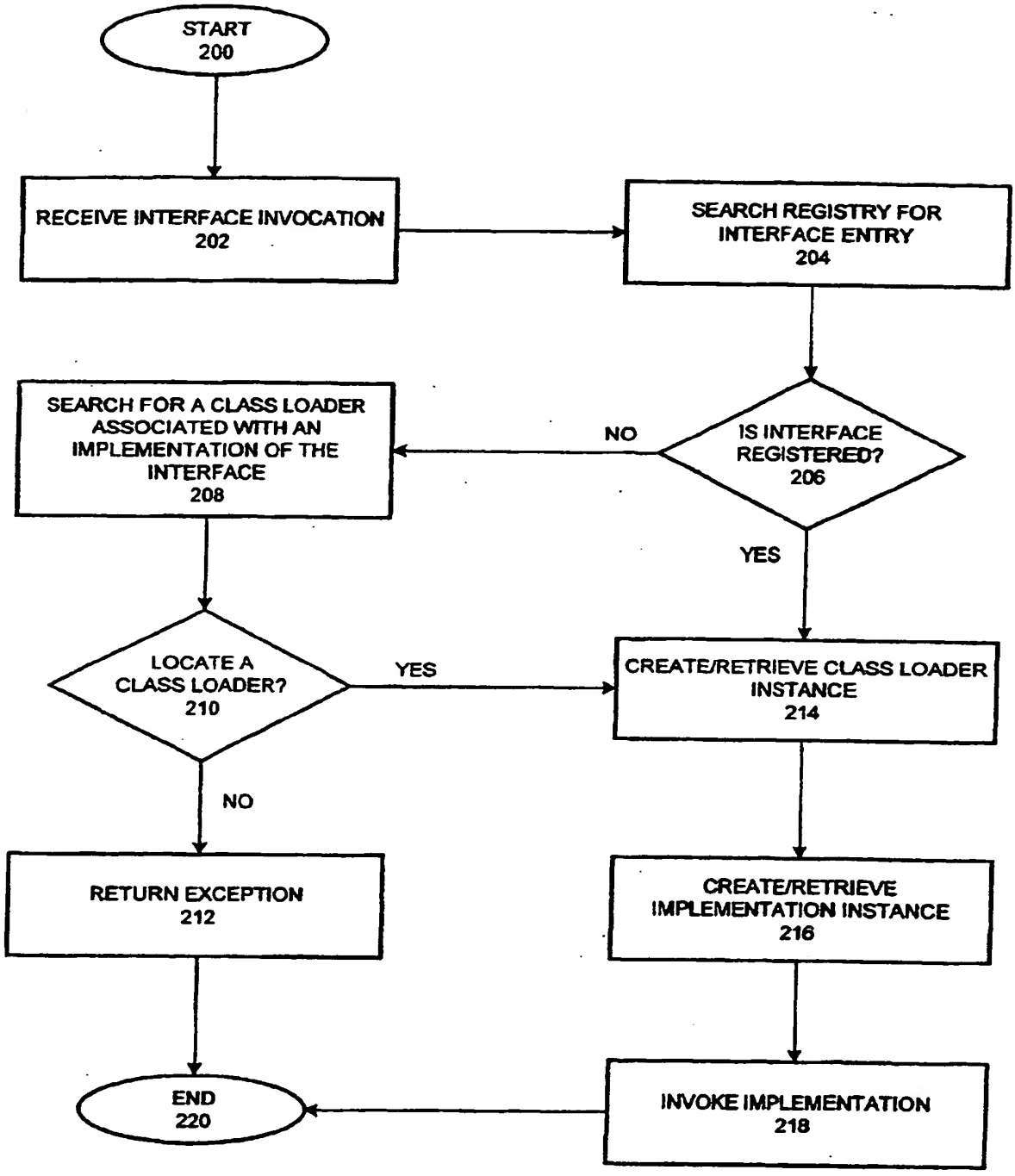
START
200
RECEIVE INTERFACE INVOCATION
202
SEARCH REGISTRY FOR INTERFACE ENTRY
204
IS INTERFACE REGISTERED?
206
NO
SEARCH FOR A CLASS LOADER ASSOCIATED WITH AN IMPLEMENTATION OF THE INTERFACE
208
YES
LOCATE A CLASS LOADER?
210
YES
CREATE/RETRIEVE CLASS LOADER INSTANCE
214
NO
RETURN EXCEPTION
212
CREATE/RETRIEVE IMPLEMENTATION INSTANCE
216
END
220
INVOKE IMPLEMENTATION
218

FIG. 2

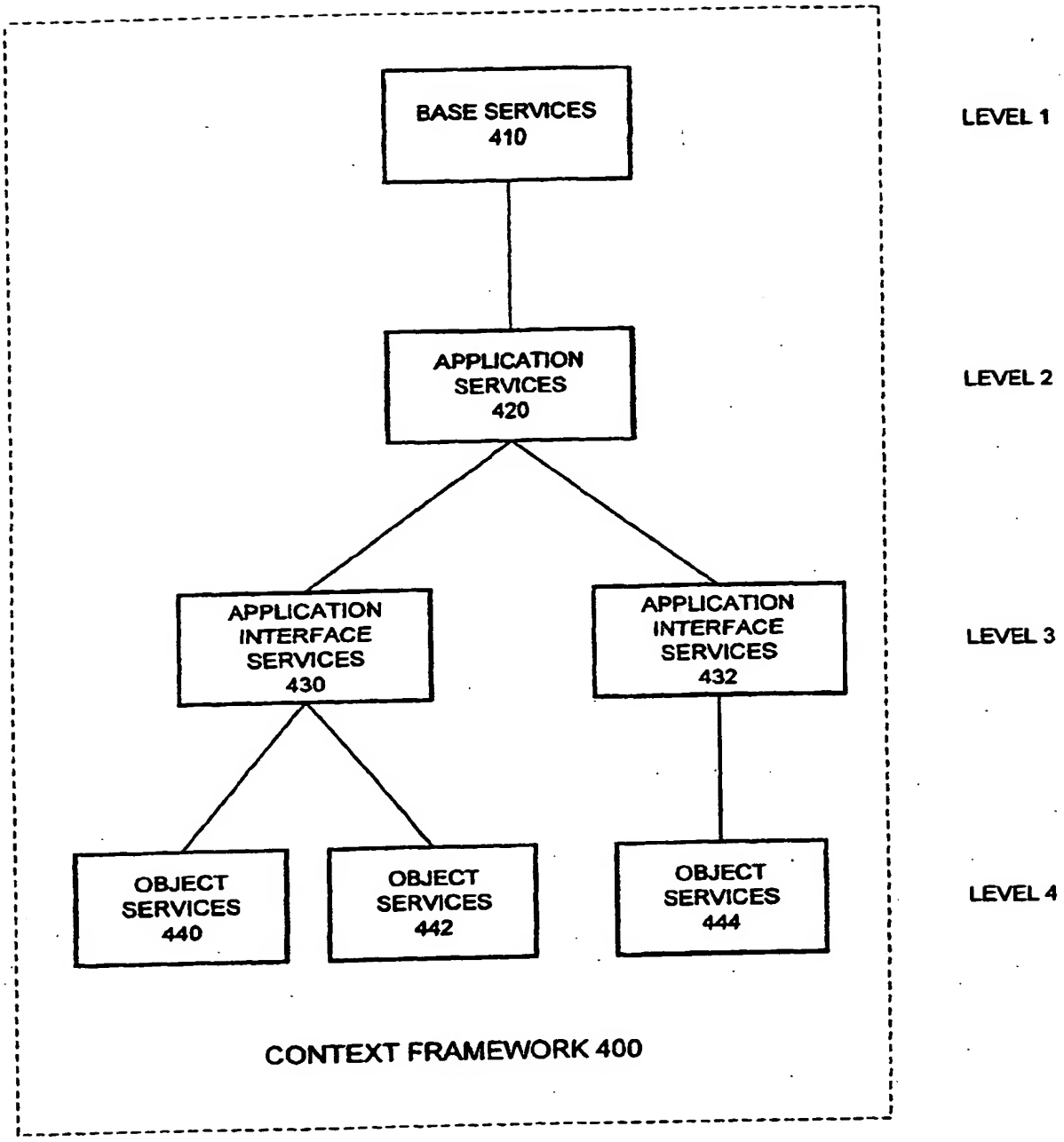
BASE SERVICES
410
LEVEL 1
APPLICATION
SERVICES
420
LEVEL 2
APPLICATION
INTERFACE
SERVICES
430
APPLICATION
INTERFACE
SERVICES
432
LEVEL 3
OBJECT
SERVICES
440
OBJECT
SERVICES
442
OBJECT
SERVICES
444
LEVEL 4
CONTEXT FRAMEWORK 400

FIG. 4